Parity Checker Example

A string of bits has “even parity” if the number of 1’s in the string is even.

• Design a circuit that accepts a bit-serial stream of bits and outputs a 0 if the parity thus far is even and outputs a 1 if odd:

```
bit stream   in  | Parity Checker  | out
          | 0 if even parity | 1 if odd parity
```

Example: In: 0 1 0 1 1 1 0 0

• Can you guess a circuit that performs this function?

Formal Design Process

• “State Transition Diagram”
  - circuit is in one of two states.
  - transition on each cycle with each new input over exactly one arc (edge).
  - Output depends on which state the circuit is in.

Example: In: 0 1 0 1 1 1 0 0

EVEN

OUT=1

Even even odd even odd even odd even

ODD

OUT=1

Odd even even odd even odd even odd

Let 0 = EVEN state, 1 = ODD state

| present state | OUT | IN  | next stat
|---------------|-----|-----|----------
| EVEN          | 0   | 0   | EVEN
| EVEN          | 0   | 1   | ODD
| ODD           | 1   | 0   | ODD
| ODD           | 1   | 1   | EVEN

Invent a code to represent states:

<table>
<thead>
<tr>
<th>present state (ps)</th>
<th>OUT</th>
<th>IN</th>
<th>next state (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Derive logic equations from table (how?):

OUT = PS
NS = PS xor IN
Formal Design Process

Logic equations from table:
OUT = PS
NS = PS xor IN

- Circuit Diagram:
  - xor gate for ns calculation
  - DFF to hold present state
  - no logic needed for output

- Review of Design Steps:
  1. Circuit functional specification
  2. State Transition Diagram
  3. Symbolic State Transition Table
  4. Encoded State Transition Table
  5. Derive Logic Equations
  6. Circuit Diagram
     FFs for state
     CL for NS and OUT

Finite State Machines (FSMs)

- Type of sequential circuit:
  - output depends on present and past inputs
  - effect of past inputs is represented by the current state

- Behavior is represented by State Transition Diagram:
  - traverse one edge per cycle.

FSM Implementation

- FFs form state register
- number of states ≤ 2^n • number of flip-flops
- CL implements calculates next state and output

Combination Lock Example

- Used to allow entry to a locked room:
  2-bit serial combination. Example 01,11:
  1. Set switches to 01, press ENTER
  2. Set switches to 11, press ENTER
  3. OPEN is asserted (OPEN=1).
     If wrong code, ERROR is asserted (after second combo word entry).
     Press Reset at anytime to try again.
Announcements

Combination Lock Example

- Used to allow entry to a locked room:
  - 2-bit serial combination. Example 01,11:
    1. Set switches to 01, press ENTER
    2. Set switches to 11, press ENTER
    3. OPEN is asserted (OPEN=1).
       If wrong code, ERROR is asserted (after second combo word entry).
       Press Reset at anytime to try again.

Symbolic State Transition Table

<table>
<thead>
<tr>
<th>RESET</th>
<th>ENTER</th>
<th>COM1</th>
<th>COM2</th>
<th>Preset State</th>
<th>Next State</th>
<th>OPEN</th>
<th>ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>START</td>
<td>START</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>*</td>
<td>START</td>
<td>BAD1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>*</td>
<td>START</td>
<td>OK1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>*</td>
<td>0</td>
<td>OK1</td>
<td>OK1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>*</td>
<td>OK1</td>
<td>BAD2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>OK1</td>
<td>OK2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>*</td>
<td>*</td>
<td>0</td>
<td>OK2</td>
<td>OK2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>*</td>
<td>*</td>
<td>BAD1</td>
<td>BAD1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>BAD1</td>
<td>BAD2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>BAD2</td>
<td>START</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Decoder logic for checking combination (01,11):
### Encoded ST Table

- **Assign states:**
  - START=000, OK1=001, OK2=011
  - BAD1=100, BAD2=101
- **Omit reset. Assume that primitive flip-flops have reset input.**
- **Rows not shown have don't cares in output. Correspond to invalid PS values.**

<table>
<thead>
<tr>
<th>NS2</th>
<th>NS1</th>
<th>NS0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### FSM Implementation Notes

- **General FSM form:**
  - inputs → CL → outputs
  - present state → next state
- **All examples so far generate output based only on the present state:**
- **Commonly name Moore Machine**

### State Encoding

- **One-hot encoding of states.**
- One FF per state.
- **Ex: 3 States**
  - STATE1: 001
  - STATE2: 010
  - STATE3: 100
- **In general:**
  - \# of possible states = \(2^N\) if \(N\) FFs
- **However, often more than log\_2(\# of states) FFs are used, to simplify logic at the cost of more FFs.**
- **Extreme example is one-hot state encoding.**

### One-hot encoded FSM

- **Even Parity Checker Circuit:**
  - FFs must be initialized for correct operation (only one 1)
  - **In General:**
  - state FF
  - input to other state FF logic and/or output
One-hot encoded combination lock